UNIVERSIDADE FEDERAL DA FRONTEIRA SUL
CAMPUS CHAPECÓ
CURSO DE CIÊNCIA DA COMPUTAÇÃO

ESTUDO COMPARATIVO ENTRE PLATAFORMAS DE DEEP LEARNING

ELIAS AUGUSTO FANK

CHAPECÓ
2018

ELIAS AUGUSTO FANK

ESTUDO COMPARATIVO ENTRE PLATAFORMAS DE DEEP LEARNING

Trabalho de conclusão de curso de graduação apresentado como requisito para obtenção do grau de Bacharel em Ciência da Computação da Universidade Federal da Fronteira Sul.
Orientador: Prof. Dr. Denio Duarte

CHAPECÓ
2018

Fank, Elias Augusto

Estudo comparativo entre plataformas de Deep Learning / por Elias Augusto Fank. – 2018.
54 f.: il.; 30 cm.

Orientador: Denio Duarte
Monografia (Graduação) - Universidade Federal da Fronteira Sul, Ciência da Computação, Curso de Ciência da Computação, SC, 2018.

1. Deep Learning. 2. Estudo Comparativo. 3. Plataformas. 4. Benchmark. I. Duarte, Denio. II. Título.

E-mail: eliasfank@hotmail.com

ELIAS AUGUSTO FANK

ESTUDO COMPARATIVO ENTRE PLATAFORMAS DE DEEP LEARNING

Trabalho de conclusão de curso de graduação apresentado como requisito para obtenção do grau de Bacharel em Ciência da Computação da Universidade Federal da Fronteira Sul.

Orientador: Prof. Dr. Denio Duarte

Aprovado em: 05\07\2018

BANCA EXAMINADORA:

Dr. Denio Duarte - UFFS

Ma. Andressa Sebben - UFFS

Dr. Guilherme Dal Bianco - UFFS

RESUMO

Técnicas de *Deep learning* vem mostrando avanços em várias tarefas de *Machine learning*. Porém a implementação dessas técnicas é muito complexa. Assim, para ajudar na implementação de projetos de *Deep learning*, *softwares* estão sendo criados. Já existe uma quantidade considerável desses *softwares* disponível, o que acaba trazendo uma dificuldade na escolha de quem procura começar um projeto. Com o objetivo de auxiliar nessa escolha, esse trabalho traz um estudo comparativo entre as plataformas *open source* e distribuídas *Apache Singa*, *Graphlab* e *H2O*. Resultados detalhados de testes com uma base de dados compostas por imagens e outra composta por atributos alfanuméricos foram produzidos. Aspectos como o tempo de treinamento, tempo de predição, utilização de recursos e acurácia dos algoritmos de cada plataforma foram avaliados.

Palavras-chave: Deep Learning. Estudo Comparativo. Plataformas. Benchmark.

ABSTRACT

Deep learning techniques has been showing advances in various Machine learning tasks. However, the implementation of these techniques is very complex. Thus, to help the implementation of Deep learning projects, software tools are being proposed. A considerable amount of these tools already exists. This leads to a difficulty on the choice of who is looking to start a project. In order, to assist in this choice, this work brings a comparative study between the open source and distributed Apache Singa, *Graphlab* and H2O platforms. Detailed test results using a database composed of images and another composed of alphanumeric attributes were produced. And aspects such as training time, prediction time, hardware resource utilization and accuracy of the algorithms of each platform were evaluated.

Keywords: Deep Learning, Comparative Study, Platforms, Benchmark.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

DL	*Deep Learning*
ML	*Machine Learning*
MLP	*Multi Layer Perceptron*
CNN	*Convolutional Neural Network*
RBM	*Restricted Boltzmann Machine*
RNN	*Recurrent Neural Network*
BP	*Back Propagation*
CD	*Contrastive Divergence*
SGD	*Stochastic Gradient Descent*
CPU	*Central Processing Unit*
GPU	*Graphics Processing Unit*
MNIST	*Database of Handwritten Digits*
KDD	*Database of Connection Examples*
MB	*Megabytes*
RAM	*Random Access Memory*

SUMÁRIO

1 INTRODUÇÃO

Algumas tarefas como a identificação de objetos em imagens, transformação de trechos de fala em texto, recomendação de produtos aos consumidores de *sites e-commerce* baseando-se em suas buscas, entre outros, são alguns dos problemas que podem ser resolvidos e explorados a partir de sistemas de *Machine learning* (ML).

Segundo (MITCHELL, 1997), *Machine learning* é uma área da computação que busca melhorar a autonomia dos computadores através da sua própria experiência. Ou seja, como fazer com que algoritmos possam aprender a partir de uma base de dados de entrada.

Levando em consideração a complexidade de tarefas que podem ser resolvidas com o uso de ML, algumas das técnicas convencionais vêm sendo substituídas por algoritmos de ML baseados em *Deep Learning*. Assim, (LECUN; BENGIO; HINTON, 2015) apontam que quando um problema é complexo, os dados disponíveis costumam estar representados de uma forma também complexa, e com algoritmos de *Deep learning* é possível automaticamente extrair e descobrir qual é a melhor representação para cada característica dos dados. Desta forma, elimina-se uma etapa inicial que é transformar os dados de entrada em estruturas pré definidas, para que os algoritmos de ML possam funcionar corretamente.

Segundo (DENG; YU, 2014), a partir de 2006, o *Deep learning* estruturado, também conhecido como *Deep learning* ou aprendizagem hierárquica, emergiu como uma nova área de pesquisa em aprendizagem de máquina. Durante os últimos anos, as técnicas desenvolvidas em pesquisas de *Deep learning* tem impactado em trabalhos de processamento de sinais e processamento de informações.

Os algoritmos que compõe essas técnicas são formados por múltiplas camadas de processamento para aprender representações de dados com vários níveis de abstração (LECUN; BENGIO; HINTON, 2015). Ou seja, os dados de entrada, como por exemplo os pixeis de uma imagem formam a primeira camada que gera as entradas para a segunda e assim por diante.

Deep learning não é uma técnica recente, pois já faz parte de várias aplicações na atualidade. Sua utilização vem se intensificando por consequência do aumento do poder computacional das máquinas (principalmente do processamento gerado por placas gráficas cada vez mais robustas), da popularização das técnicas para criação de modelos baseados em *Deep Learning* e do tamanho dos dados utilizados para treinamento (DENG; YU, 2014).

O uso crescente dessas técnicas baseadas em *Deep Learning* faz com que *softwares*

sejam criados para auxiliar no desenvolvimento de sistemas que requerem o uso de algoritmos de *Deep Learning*. Esses *softwares* permitem um desenvolvimento e uma implementação mais eficiente (BAHRAMPOUR et al., 2015). Existem hoje diversos tipos de ferramentas para auxiliar no desenvolvimento, bem como *frameworks* (*softwares* com bibliotecas previamente desenvolvidas) e também plataformas (*softwares* mais robustos, capazes de abstrair grande parte do desenvolvimento). Como exemplos de plataformas que facilitam a criação de projetos de *Deep Learning* pode-se citar: *Singa*, *H2O* e *GraphLab*, quais serão objetos de estudo neste trabalho.

1.1 Objetivos

1.1.1 Geral

Comparar plataformas que oferecem suporte para o desenvolvimento de modelos de *Deep Learning*, identificando as suas características em termos de desempenho computacional (utilização de recursos, tempos de treinamento e execução) e dificuldade de implementação.

1.1.2 Específicos

- Relacionar algoritmos e arquiteturas de *Deep Learning* já estabilizados no campo de pesquisa;
- Relacionar as mais conceituadas plataformas de *Deep Learning*;
- Definir os conjuntos de dados para avaliar as plataformas;
- Definir ambientes de testes para os experimentos de avaliação;
- Selecionar as métricas para avaliação das plataformas;
- Avaliar e comparar os resultados obtidos nos testes executados.

1.2 Justificativa

Com a maior disponibilidade de recursos computacionais, o uso de *Deep learning* vem se intensificando. Isso porque diferentes campos de pesquisa como biomedicina, análise de imagem, análise de mídias sociais, entre outros começam a obter sucesso ao fazer o uso de *Deep*

learning (SHAMS et al., 2017). Com esse estímulo ao desenvolvimento, aumenta o interesse em se desenvolver soluções para atender a carga de trabalho em *Deep learning*.

Algoritmos de *Deep learning* vem se destacando ao mostrar resultados práticos com um bom desempenho quando grandes modelos com mais parâmetros são necessários (DEAN et al., 2012). Porém como desvantagem, os algoritmos de treinamento para modelos baseados em *Deep learning* costumam exigir muitos recursos de *hardware*, necessitando ajustes finos na parametrização (UETZ; BEHNKE, 2009). Esses ajustes geralmente são complexos e apenas usuários especialistas, como engenheiros ou cientistas, conseguem obter sucesso nessa prática. Em decorrência disso, surge uma demanda na utilização de ferramentas que auxiliem na construção de modelos baseados em *Deep learning*. Com tais ferramentas, usuários sem conhecimentos sólidos em parametrização de modelos de *Deep learning* podem criar seus próprios projetos.

Com mais *softwares* sendo desenvolvidos, o "leque" de opções disponíveis para escolher, ao se iniciar um projeto, aumenta. Sendo assim, fica difícil para o projetista saber qual é o melhor *software* a ser utilizado em determinadas situações. Comparações entre esses *softwares* são necessárias para ajudar na decisão, uma vez que possibilitam uma classificação entre eles.

Para exemplificar, um usuário pode estar interessado em modelos com melhores resultados de aprendizagem (melhor acurácia). Outro pode estar interessado em modelos com o menor tempo de treinamento. Ou ainda, pode haver a necessidade de uma ferramenta que tenha um melhor gerenciamento de *hardware* quando o mesmo se encontra com certa limitação.

Outro benefício de um estudo comparativo entre *softwares* surge quando os mesmos são expostos aos mais variados testes. Pelo fato deles serem novos e muitos ainda estarem em desenvolvimento, sua estrutura não está consolidada. Assim, pontos a melhorar podem ser destacados nos *softwares* em estudo.

Em resumo, para comparar de forma mais eficaz a variedade de ferramentas de *software* de DL, eles devem ser comparados com diferentes métricas ao longo de diferentes ambientes de configuração. E, neste trabalho, será apresentado um estudo onde cada um dos *softwares* será submetido a testes com diferentes bases de dados e diferentes configurações de execução.

Dado o contexto apresentado acima, isto é, a variedade de ferramentas disponíveis para aplicação de *Deep learning*, a complexidade na configuração e no uso de algoritmos de *Deep learning* e a variedade de ferramentas disponíveis, este projeto objetiva classificar plataformas de propósitos semelhantes de acordo com métricas sugeridas.

1.3 Estrutura do Trabalho

O restante do trabalho está organizado da seguinte forma. O próximo capítulo traz uma revisão sobre o que é *Deep learning* e dois exemplos de rede neural. Depois, no terceiro capítulo, estão apresentadas as plataformas que são o foco de estudo deste trabalho. O quarto capítulo traz uma relação com 3 trabalho correlatos. O quinto traz os experimentos; nele são apresentadas as bases de dados utilizadas no testes, ambiente de configuração, os experimentos realizados e os resultados obtidos. Por fim, o capítulo que traz a conclusão deste trabalho.

2 DEEP LEARNING

Deep learning é uma classe de técnicas de *Machine learning* que explora o processamento de informações em níveis não lineares. Essas técnicas são formadas por estruturas divididas em camadas (geralmente redes neurais), onde cada camada representa um conjunto de representações ocultas dos dados e muitas vezes complexas para o entendimento humano (DENG; YU, 2014).

Na Figura 2.1 (exemplo de reconhecimento de imagem) pode-se observar diagramaticamente como uma rede de *Deep learning* funciona. Na primeira camada tem-se a representação de características bem simples da imagem, neste caso, traços retos e demais formas geométricas. Na medida em que se avança pelas camadas, essas características simples se unem constituindo representações mais complexas. Então, ao chegar na camada de saída da rede um rosto é identificado, neste caso, pertencente a "Sara".

Redes neurais convolutivas e *Multilayer Perceptron* (MLP) são as principais redes de *Deep learning* no sentido de já estarem estabilizadas no campo de pesquisa em *Deep learning* (AREL; ROSE; KARNOWSKI, 2010). Por isso, redes MLP são as redes mais comumente encontradas em ferramentas de *Deep learning*. Como pode ser observado na Tabela 3.1, as plataformas que serão alvo de análise neste trabalho tem a implementação da rede MLP em comum. A rede MLP e a rede neural convolutiva, com características para tarefas supervisionadas, serão apresentadas nas próximas seções.

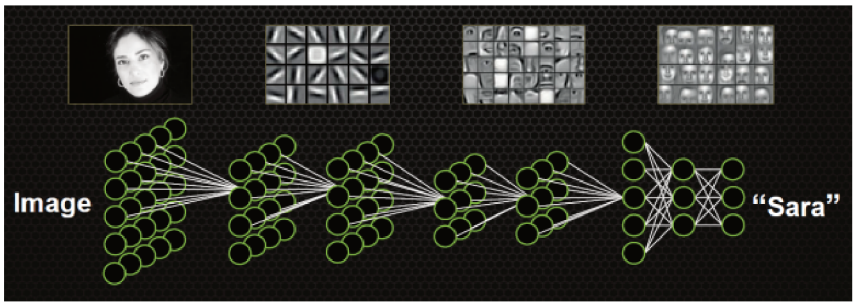

Figura 2.1 – Representação de um dado em uma rede de *Deep learning*

Fonte: Adaptado de *devblogs.nvidia.com*

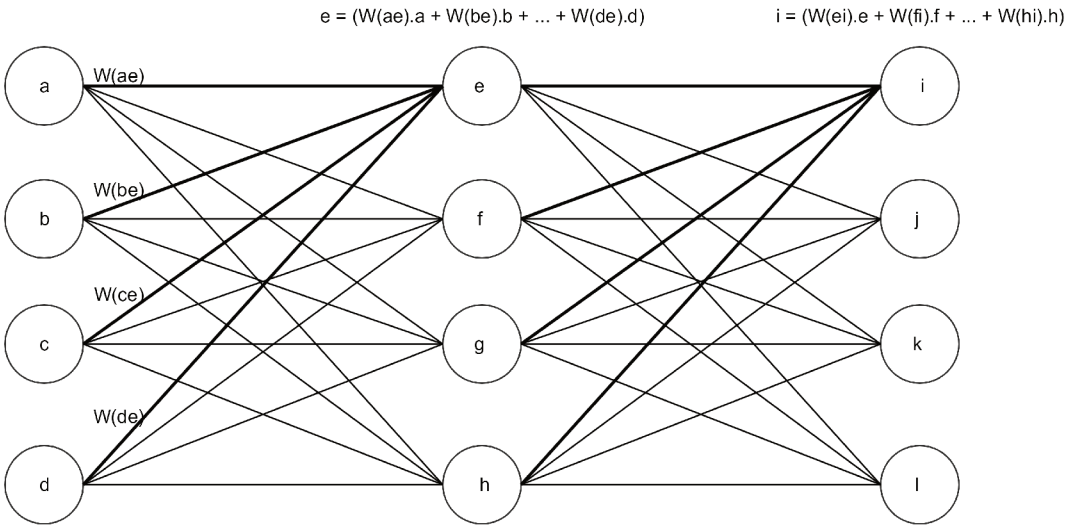

Figura 2.2 – *Feed forward Neural Network*

2.1 Redes Multilayer Perceptron (MLP)

Uma rede MLP é formada basicamente por uma rede neural. Redes Neurais são modelos ou estruturas de dados constituídas de neurônios artificiais. Um neurônio recebe como entrada vários valores e geralmente computa um único valor de saída utilizando uma função de ativação (funções de ativação serão apresentadas na subseção 2.1.1). O formato mais simples de redes neurais são as *Feed Forward Neural Networks*. Este tipo de rede é constituído de camadas, e todos os neurônios de uma camada são conectados com os neurônios da próxima, como pode ser visto na Figura 2.2. Algumas redes utilizam um valor b (bias) que é somado ao valor de cada nó, esse elemento permite que a rede se adapte melhor, em alguns casos, aos dados que lhe são passados.

As *Feed Forward Neural Networks* possuem dois algoritmos principais, o *forward propagation* (que será apresentado na Subseção 2.1.2) e o *back propagation* (que será apresentado na Subseção 2.1.3). Eles são usados para computar o vetor de saída de ativação da rede e reajustar os parâmetros da rede com base em alguma medida de erro, respectivamente. Com exceção das camadas de entrada e saída, todas as camadas da *Feed forward Neural Network* são chamadas camadas escondidas (*hidden layers*).

2.1.1 Função de ativação

Funções de ativação são funções não lineares localizadas em cada neurônio e são responsáveis por "ativar" cada respectivo neurônio quando este representa fortemente (limiar ajustável) alguma característica do dado de entrada.

As principais funções de ativação utilizadas são:

- Sigmoid: essa função não linear tem a seguinte forma matemática:

$$\sigma(x) = \frac{1}{(1 + \mathrm{e}^{-x})} \tag{2.1}$$

  e sua forma geométrica pode ser observada na Figura 2.3 a. Basicamente, quando o valor computado pelo neurônio tende à infinito negativo a saída da função é um valor que se aproxima de zero. E quando o valor computado pelo neurônio é um número positivo que tende ao infinito, o resultado da função de ativação se aproxima de 1.

- Tanh: essa função não linear pode ser escrita matematicamente da seguinte forma:

$$tanh(x) = \frac{sinh(x)}{cosh(x)} = \frac{\frac{\mathrm{e}^{x} - \mathrm{e}^{-x}}{2}}{\frac{\mathrm{e}^{x} + \mathrm{e}^{-x}}{2}} = \frac{\mathrm{e}^{2x} - 1}{\mathrm{e}^{2x} + 1} \tag{2.2}$$

  e sua representação geométrica está na Figura 2.3 b. É basicamente uma função *sigmoid* modificada que responde melhor a pequenas variações nos valores computados pelo neurônio.

- Relu (Rectified linear unit): sua forma matemática pode ser vista como:

$$relu(x) = \begin{cases} 0 & \text{se } x < 0 \\ x & \text{caso contrário} \end{cases} \tag{2.3}$$

  e seu gráfico está na Figura 2.3 c. A Relu é uma função bastante simples: valores menores que 0 são retificados para 0, caso contrário, serão os próprios valores computados. O custo computacional da Relu é bastante baixo em relação às anteriores e, assim, vem sendo cada vez mais como função de ativação em redes neurais.

- Softmax: essa função é utilizada em algumas redes quando o objetivo é classificar um entrada para duas ou mais classes. Geralmente aplicada na última camada da rede, a Softmax retorna valores entre 0 e 1. Assim para uma entrada qualquer, o vetor resultante da última camada da rede, depois de aplicada a função *Softmax*, terá em cada posição a probabilidade daquela entrada pertencer à cada classe. Forma matemática da função *Softmax*:

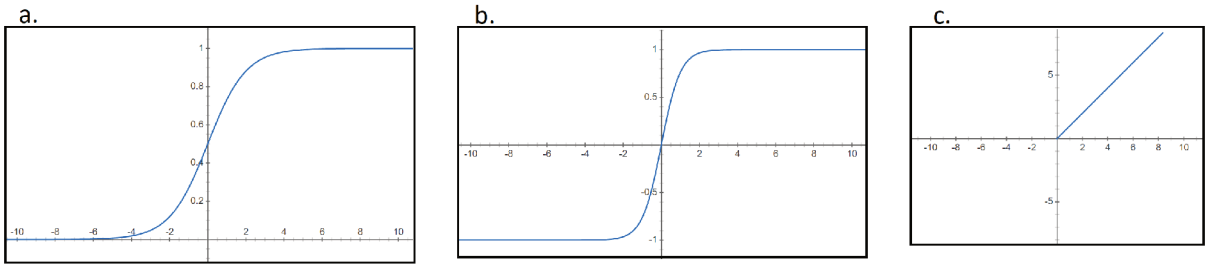

Figura 2.3 – Funções de ativação

$$softmax(x) = \frac{e_i^x}{\sum_{j=1}^{J} e^{xj}}, \text{para } i = 1 \text{ até nº classes} \tag{2.4}$$

2.1.2 Forward Propagation

Essa etapa é responsável, como o nome já sugere, de propagar os dados da camada de entrada até a camada de saída. Cada camada da rede computa sua saída com uma função de ativação gerando um vetor de valores. Cada elemento deste vetor é então multiplicado por um peso (w) presente em cada aresta que liga dois neurônios. A Figura 2.2 apresenta o papel do peso nas interligações entre as unidades da rede. A Equação 2.5 apresenta a definição matemática do papel do peso na atualização das camadas.

$$l_k = \sum_{i=1}^{n} l_{k_i} w_i \tag{2.5}$$

Onde l_k é a camada a ser atualizada.

2.1.3 Back Propagation

Essa etapa é a responsável pela otimização do modelo a ser aprendido. Os valores das unidades são ajustados para uma nova rodada de *forward propagation*. Assim, neste processo os valores dos *weights* (arestas da rede) são atualizados. Para atualizar esse valores, a rede, depois de fazer o processo de *forward propagation*, compara a saída com o valor ideal (rótulo para esta entrada). A diferença entre valores caracteriza o erro E. Baseado no erro é conhecida a variação $\triangle$ necessária que precisa ser aplicada em W (*weights*) para chegar a saída desejada. Assim, os pesos são atualizados da saída em direção à entrada, movimento que define o nome da técnica: *back propagation*, ou seja, propagação para trás. O erro E encontrado na saída volta atualizando as camadas internas da rede, que em seguida, realiza o *forward propagation*. Assim, os valores

das camadas l_k atualizam os valores das camadas l_{k-1}. Esse processo ocorre até ser encontrado um *E* que atende um limiar definido ou quando se estabiliza.

Mesmo possuindo apenas o erro da última camada, como as funções de ativação f são deriváveis, é utilizada a função que deriva o erro com relação a cada parâmetro. Então para uma camada *n* de uma rede neural temos a Equação 2.6 para computar a variação de *W*.

$$\Delta W = -\frac{\partial E}{\partial W} \tag{2.6}$$

Assim, depois de aplicar a variação necessária em *W*, pode-se calcular o erro para a camada *n-1* e ajustar novamente os valores de *W* até chegar na camada de entrada.

2.2 Redes Neurais Convolutivas (CNN)

Uma CNN é utilizada principalmente para tratar problemas onde os dados podem ser estruturados em duas dimensões (AREL; ROSE; KARNOWSKI, 2010). No caso de um problema onde uma imagem é analisada, a ideia principal é fazer a análise de uma pequena porção da imagem independentemente da sua posição e independentemente do restante dela. Para isso a imagem é quebrada em partes. Cada parte da imagem pode então ser analisada e também pode ser combinada com as outras criando representações mais abstratas da imagem.

Da mesma forma como as MLP, uma CNN é formada por uma rede neural. Mas sua principal característica é fazer dois processos que antecedem a entrada das características na rede. Isso para atingir os objetivos descritos no parágrafo anterior. Esses processos são chamados de convolução e subamostragem (GOODFELLOW; BENGIO; COURVILLE, 2016). Nas subseções seguintes esses dois processos serão descritos levando-se em consideração um problema de análise de imagem.

2.2.1 Convolução

O principal propósito deste processo é extrair informações da imagem. A convolução preserva a relação espacial entre os pixeis criando mapas de características. Cada um desses mapas representa um filtro aplicado sobre a imagem. O filtro mais comumente aplicado é o que faz pequenas amostragens em forma de quadrados. Como pode ser observado na Figura 2.4, um dos mapas de características é criado a partir do movimento (*slide*) de tamanho 4x4 (pixeis) na imagem.

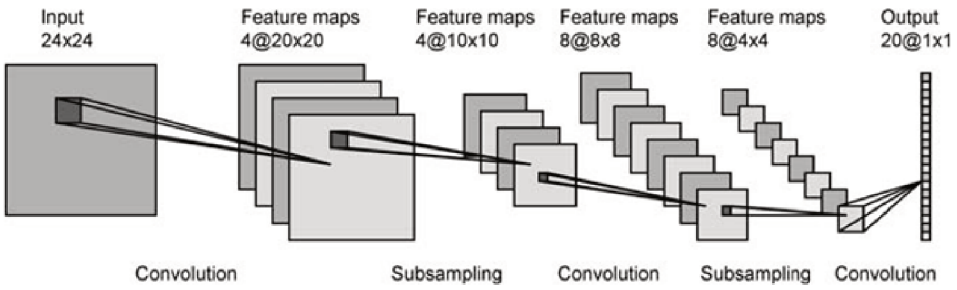

Figura 2.4 – Etapas da rede neural convolutiva

Fonte: Adaptado de *www.researchgate.net*

2.2.2 Subamostragem

Neste processo, o objetivo é reduzir o tamanho dos mapas de características. Na maioria das vezes isto é feito a partir da redução de pixeis vizinhos para um único valor. A forma mais comum de fazer essa redução é utilizar o maior valor dos pixeis na vizinhança, ou fazer a média entre eles. Na Figura 2.4, pode-se observar que na primeira aplicação de subamostragem, cada agrupamento de 2x2 pixeis gerou um pixel. Assim, cada mapa de característica teve seu tamanho reduzido pela metade (de 4 mapas com dimensão 20x20, para 4 mapas com dimensão 10x10).

2.3 Rede selecionada

Esse capítulo apresentou os conceitos básicos de *Deep learning*, bem como as técnicas MLP e CNN, que já são amplamente utilizadas em problemas de ML (AREL; ROSE; KARNOWSKI, 2010). Dentre as técnicas apresentadas e as disponíveis atualmente, a MLP foi utilizada para a execução de testes deste trabalho quando se fez necessária a parametrização manual das plataformas. A definição por executar os testes em uma rede MLP se deu pelo fato de ser a rede implementada nas três plataformas que serão avaliadas. O próximo capítulo apresenta cada uma das plataformas selecionadas.

3 PLATAFORMAS SELECIONADAS

Com o intuito de auxiliar a elaboração de um projeto de *Deep learning*, nos últimos anos, estão surgindo ferramentas que trazem consigo implementações de diferentes famílias de redes neurais para este tipo de abordagem. Assim, este trabalho propõe um estudo comparativo de algumas destas ferramentas a fim de identificar o desempenho sob algumas métricas a serem definidas. Porém, dentre toda a gama de ferramentas disponíveis, foram selecionadas aquelas que possuem implementações completas de redes de *Deep learning* e não tenham custo de aquisição. Sendo assim, as seguintes plataformas foram avaliadas neste trabalho: *Apache Singa* , *H2O* e *GraphLab*. As mesmas serão apresentadas nas próximas seções.

3.1 Apache Singa

Singa é uma plataforma de *Deep learning* distribuída (OOI et al., 2015; WANG et al., 2015), que traz modelos como redes neurais convolutivas (CNN), máquinas de Boltzmann restritas (RBM) e redes neurais recorrentes (RNN). Essa plataforma pode "rodar" *frameworks* para treinamento síncrono, assíncrono e híbrido. O treinamento síncrono melhora a eficiência de uma iteração, e o treinamento assíncrono melhora a taxa de convergência. Usuários desta plataforma também podem escolher rodar um *framework* híbrido para ter um equilíbrio entre eficiência e taxa de convergência.

Apache Singa é um software de código aberto (disponível em *github.com*), e está sob a licença "Apache 2.0". Os seguintes componentes fazem parte da plataforma:

- *Worker* e *Server*: são arquiteturas lógicas criadas para organização e que facilitam a distribuição. Um *Worker* executa as atualizações dos parâmetros da rede. E um *Server* mantem as informações atualizadas e as distribui para cada *Worker*.

- *TrainOneBatch*: essa é uma função que é "chamada" por cada *Worker* em cada iteração do algoritmo SGD (algoritmo responsável principalmente por calcular o gradiente descendente e encontrar o mínimo local para uma função). Ela executa basicamente duas funções: a de *back-propagation* (BP) e a de *contrastive divergence* (CD). CD é usado apenas em alguns modelos de *Deep learning*.

O algoritmo SGD é utilizado nesta plataforma para treinar os modelos de *Deep learning*. A carga de processamento é dividida para cada *Worker* e cada *Server* como na Figura 3.1.

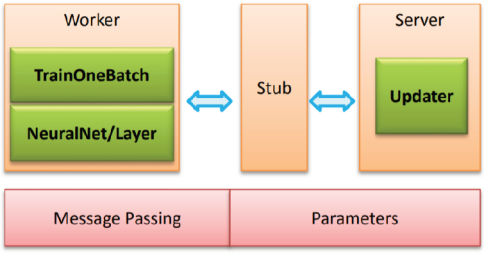

Figura 3.1 – Abstração da arquitetura da plataforma *Singa*

Fonte: (OOI et al., 2015)

Em cada iteração, cada *Worker* chama a função *TrainOneBatch* para computar os gradientes do algoritmo SGD. A função *TrainOneBatch* utiliza um objeto chamado de *NeuralNet* que representa uma rede neural para obter as informações das camadas e calcular os gradientes. Quando os gradientes são devidamente calculados, eles são enviados para o componente *Stub* que é responsável por gerenciar todos as requisições e encaminhar ao devido *Server* para a atualização. A partir daí o *Server* encaminha as informações atualizadas para cada *Worker* utilizando também o componente *Stub*.

3.2 H2O

A plataforma *H2O* diferentemente da plataforma *Singa* vem com um propósito mais geral para ML. Sob a licença "Apache 2.0", possui um código *Open Source*.

Segundo (CANDEL et al., 2015), utilizando compressão de memória, *H2O* consegue trabalhar com um grande volume de dados na memória, mesmo em um *cluster* pequeno. Essa plataforma também pode ser configurada para trabalhar em uma estrutura *multi-cluster* e disponibiliza interface para várias linguagens como R, Python, Scala, Java, JSON e CoffeeScript/JavaScript.

H2O traz implementações de algoritmos de *Deep learning*, que é o foco do comparativo deste trabalho. Apesar disso, traz outros algoritmos como alguns de ML, como modelos lineares gerais (regressão linear, regressão logística), clusterização *k-means*, entre outros.

Ao utilizar essa plataforma para um problema de *Deep learning*, de maneira distribuída (*multi-cluster*), cada nó da configuração opera em paralelo com seus dados locais. Como pode ser observado na Figura 3.2, os dados de treinamento são primeiramente distribuídos para cada nó. Feito isso, a cada iteração, o nó faz uma cópia dos parâmetros globais (informações dos outros nós) e computa os parâmetros de sua rede individualmente.

Parallel distributed and multi-threaded training with SGD in H2O Deep Learning

1. Initialize global model parameters W, B
2. Distribute training data $\mathcal{T}$ across nodes (can be disjoint or replicated)
3. Iterate until convergence criterion reached:
   - 3.1. For nodes n with training subset $\mathcal{T}_n$, do in parallel:
     - a. Obtain copy of the global model parameters W_n, B_n
     - b. Select active subset $\mathcal{T}_{na} \subset \mathcal{T}_n$ (user-given number of samples per iteration)
     - c. Partition $\mathcal{T}_{na}$ into $\mathcal{T}_{nac}$ by cores n_c
     - d. For cores n_c on node n, do in parallel:
       - i. Get training example $i \in \mathcal{T}_{nac}$
       - ii. Update all weights $w_{jk} \in W_n$, biases $b_{jk} \in B_n$

$$w_{jk} := w_{jk} - \alpha \frac{\partial L(W,B|j)}{\partial w_{jk}}$$

$$b_{jk} := b_{jk} - \alpha \frac{\partial L(W,B|j)}{\partial b_{jk}}$$

   - 3.2. Set $W, B :=$ Avg$_n$ W_n, Avg$_n$ B_n
   - 3.3. Optionally score the model on (potentially sampled) train/validation scoring sets

Figura 3.2 – Treinamento paralelo distribuído e *multi-thread* da plataforma H2O

Fonte: (CANDEL et al., 2015)

3.3 GraphLab

Essa plataforma, dentre as três, é a de propósito mais geral, tanto dentro do assunto de algoritmos de ML quanto para visualização e análise de dados (LOW et al., 2012, 2014). Ainda assim traz implementações de *Deep learning*. Foi criada em cima de uma estrutura construida na linguagem C++ e possui biblioteca disponível em Python. Na documentação em seu site são encontrados os seguintes tópicos sobre o que é possível fazer usando essa plataforma:

- Analisar terabytes de dados em tempo interativo;
- Gerenciar dados tabulares, em forma de grafos, texto e imagens;
- Utilizar implementações de algoritmos de ML, incluindo *Deep learning*;
- Executar o mesmo código em uma máquina ou em um sistema distribuído, utilizando *Hadoop Yarn* ou *EC2 cluster*;
- Visualizar dados para exploração ou monitoramento de produção.

A ferramenta *Graphlab* tem uma estrutura formada basicamente por cinco componentes lógicos (DEAN et al., 2012). O primeiro, *Data graph*, como o nome já diz é um grafo e é utilizado para guardar o estado do programa, como informações do usuário e a execução atual. O segundo componente é o *Update function*, que é um procedimento sem estados que modifica os dados. O próximo componente é o chamado *Scheduling primitives* e sua função é manter a

ordenação da execução da ferramenta (seu trabalho depende muito de como são os dados que a ferramenta recebe). O quarto componente é o *Data consistency model*, que é responsável por impedir que o cálculo de cada função independente sobreponha os dados de outras (fator muito importante para manter a característica de paralelismo da ferramenta *Graphlab*). O último componente é o *Sync mechanism*, que como o nome já diz faz a sincronização. Essa sincronização é importante principalmente para o caso de utilizar-se a ferramenta de maneira distribuída, tanto em várias GPUs, quanto no caso de *multi-cluster*.

3.4 Análise geral das plataformas

Com a apresentação das plataformas, percebe-se que há algumas semelhança entre elas. A Tabela 3.1 apresenta algumas características de cada plataforma. Assim, é possível identificar quais são as redes em comum que estão presentes tanto em *Graphlab*, *H2O* e *Apache Singa*. As redes marcadas com * estão disponíveis na respectiva plataforma apenas com uso de bibliotecas de terceiros, por isso, a rede em comum que será utilizada neste trabalho é a MLP.

Outra característica que traz um aspecto em comum entre nas três plataformas é a API de programação. Assim todo o desenvolvimento (programação) deste trabalho foi realizado utilizando-se *Python*, que é a linguagem disponível em todas as plataformas.

Plataforma	**Redes implementadas**	**API disponível**	**Interface**
Singa	MLP, CNN, RBM, RNN	Python e C++	Linha de comando
H2O	MLP, CNN*, RNN*	Python, R, Java e Scala	UI
Graphlab	MLP, CNN	Python	UI

Tabela 3.1 – Comparativo das plataformas *Graphlab*, *H2O* e *Apache Singa*

4 TRABALHOS RELACIONADOS

Foram selecionados três trabalhos que tem objetivos semelhantes aos objetivos deste trabalho (comparar ferramentas de *Deep learning*). Dois dos trabalhos fazem comparações entre *frameworks* de *Deep learning*. Tais frameworks implementam funções que ajudam na construção de uma rede de *Deep learning*. O outro faz a comparações entre plataformas de *Deep learning* e é mais similar ao presente trabalho.

A primeira comparação foi proposta em (BAHRAMPOUR et al., 2015) e teve como objetivo principal comparar os *softwares*: *Caffe*, *Neon*, *TensorFlow*, *Theano* e *Torch*. O estudo compara a utilização de *hardware* e velocidade de execução dos algoritmos. Esses resultados são de dados coletados em testes executados em *frameworks* que foram "rodados" em uma máquina (*single cluster*) tanto na configuração para CPU quando para GPU.

Para avaliar os *frameworks* os seguintes aspectos foram analisados:

- Generalização: sua capacidade de incorporar diferentes tipos de arquiteturas de *Deep learning* (redes neurais Convolutivas, redes neurais Recorrentes, redes neurais *fully-connected*), diferentes procedimentos de treinamento (não supervisionado, supervisionado) e diferentes tipos de algoritmos convolutivos (exemplo: algoritmo FFT);

- Utilização de *hardware*: sua eficácia em utilizar os recursos de *hardware* tanto da CPU quanto na GPU;

- Velocidade: sua performance tanto no tempo de treinamento quanto no tempo de predição. Neste caso, as métricas utilizadas foram: *Forward Time* (o tempo que leva para uma entrada de uma determinada base de dados percorrer totalmente uma determinada rede até a saída correspondente, métrica importante para prever o tempo dessa rede ao ser utilizada em um ambiente real) e *Gradient Computation Time* (o tempo que leva para obter o gradiente de cada parâmetro de uma rede para uma determinada base de entrada).

As bases de dados utilizadas para a realização dos testes neste trabalho foram *MNIST* (LECUN; CORTES, 2010), *ImageNet* (DENG et al., 2009) e a base *IMDB* (MAAS et al., 2011).

Como conclusões, esse trabalho traz que:

- *Theano* e *Torch* são os mais generalizadores, tanto tendo maior suporte para mais diferentes arquiteturas de *Deep learning*, como possuindo maior quantidade de bibliotecas;

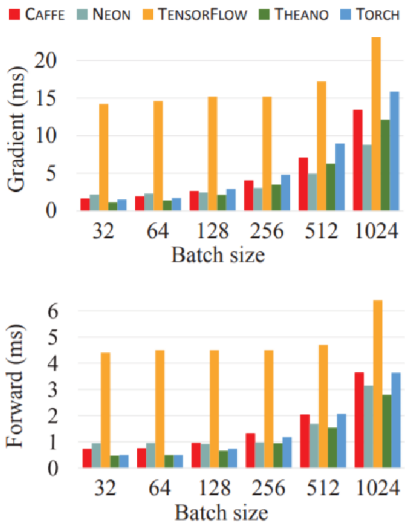

Figura 4.1 – Tempos de processamento médio para um rede LeNet (CNN) executada sobre GPU

Fonte: (BAHRAMPOUR et al., 2015)

- Para treinar e executar redes na CPU, *Torch* foi o melhor, seguido por *Theano*. *Neon* teve o pior desempenho em CPU;
- Para execução de redes convolutivas e *fully-connected* na GPU, *Torch* foi o melhor, seguido por *Theano*;
- Para treinar uma rede convolutiva e *fully-connected* na GPU, foi percebido que *Theano* é o mais rápido para pequenas redes e *Torch* é o mais rápido para redes grandes. *Neon* tem grande potencial para grandes redes convolutivas treinadas utilizando-se GPU;
- Para treinamento e execução de redes recorrentes na GPU, *Theano* é a melhor escolha;
- *Torch* tem mais documentação e melhores ferramentas para *debugging*;
- *TensorFlow* é um *framework* bastante flexível. Porém sua performance, em tempo de treinamento, utilizando uma única GPU não é muito boa se comparado a outros *frameworks* estudados neste trabalho.

O trabalho também apresentou os tempos de execução dos *frameworks* em relação a execução da otimização pelo gradiente descendente e *feed forward*. A Figura 4.1 apresenta os resultados. Perceba que o *framework TensorFlow* apresenta tempo similares independentemente do tamanho do *batch* para o treinamento, mas sempre com o pior desempenho se comparados aos outros *frameworks*.

Em (KOVALEV; KALINOVSKY; KOVALEV, 2016) é apresentado um estudo comparativo entre os seguintes *frameworks*: *Theano*, *Torch*, *Caffe*, *TensorFlow* e *DeepLearning4J*, trazendo uma avaliação quantitativa dos tempos de treinamento e predição destes *frameworks* para o problema escolhido.

Diferentemente do trabalho anterior, este concentra-se apenas em fazer o estudo para redes com arquitetura *fully-connected*, variando a largura e a profundidade da rede, também fazendo comparativos quando varia-se a função de ativação (RELU versus TANH).

A base de dados utilizada para os testes neste trabalho foi a *MNIST* (LECUN; CORTES, 2010), que é uma base de dados com imagens de dígitos escritos a mão.

As métricas utilizadas para avaliar e classificar os *frameworks* foram:

- Tempo de convergência: basicamente é o tempo que a rede leva para ser treinada;
- Tempo de predição, ou de classificação: é o tempo que, para dada uma entrada de teste, o *software* levou para percorrer toda a rede e encontrar a devida saída para essa entrada.
- Acurácia na classificação. É a porcentagem de acerto ou de erros que o *software* teve para uma determinada arquitetura implementada.
- A complexidade do *software*: é o tamanho do código, em linhas, que foi necessário para cada algoritmo nos testes.

Um dos resultados do trabalho é a constatação de que o *framework Deeplearning4J* tem um desempenho inferior para treinamento, mesmo com uma variação na profundidade da rede, como pode ser observado na Figura 4.2. Essa figura apresenta a variação do tempo de treinamento quando se altera a profundidade da rede, ou seja, configurando a rede para aumentar o número de camadas. Na figura pode-se observar também que esse resultado se mantém mesmo quando a função de ativação (em a. utiliza-se a função de ativação ReLU e em b. a função de ativação *Tanh*) é alterada.

Como mencionado acima, um dos objetivos dos autores era fazer a análise da complexidade do *software*. E para mostrar isso esse trabalho traz a Figura 4.3, que mostra quantas linhas foram necessárias para a implementação do algoritmos nos diferentes *softwares*. Para esta última comparação, os autores ressaltam que vale lembrar que diferentes linguagens de programação são utilizadas para implementação em cada *software*, o que pode influenciar na

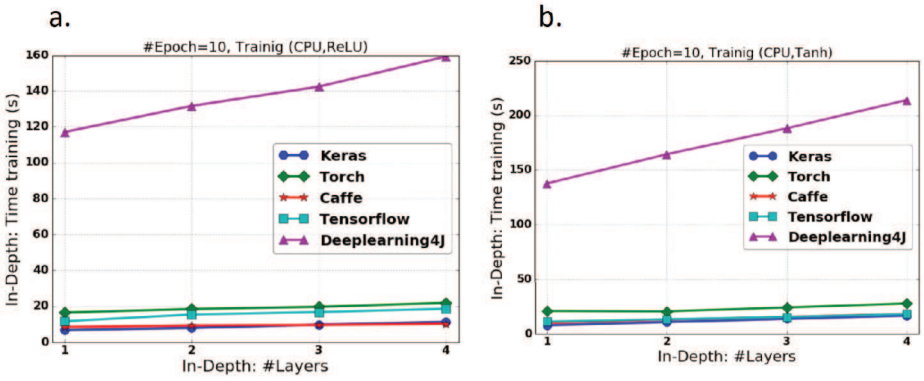

Figura 4.2 – Tempos de treinamento para uma rede neural *fully-connected* com variação de profundidade

Fonte: (KOVALEV; KALINOVSKY; KOVALEV, 2016)

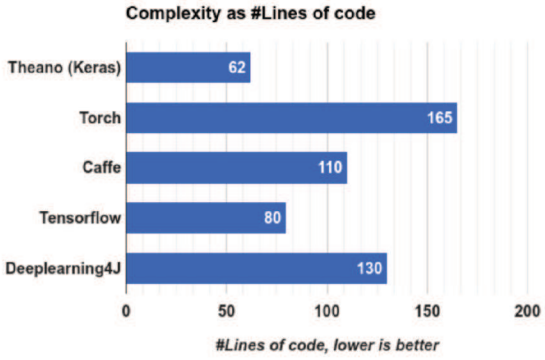

Figura 4.3 – Quantidade de linhas de código necessárias para implementação do algoritmo em cada *software*

Fonte: (KOVALEV; KALINOVSKY; KOVALEV, 2016)

quantidade de código necessário. Assim, como o mais complexo, aparece o *framework Torch*, com 165 linhas de código. O mais simples seria *Theano*, que necessitou de apenas 62 linhas.

O terceiro e último artigo é proposto por (NG et al., 2016). Este trabalho apresenta uma avaliação para as plataformas de *Singa* e *H2O* comparando a acurácia e o tempo de treinamento.

Segundo (NG et al., 2016), o grupo desenvolvedor da plataforma *Singa* publicou um comparativo de performance, comparando duas plataformas diferentes (*Singa* e *H2O*). Após a publicação do comparativo, o grupo desenvolvedor da plataforma *H2O* publicou que não era possível replicar esses mesmos resultados e assim produziu um guia de como melhorar a performance de velocidade e acurácia no *H2O*. Em contrapartida, o grupo desenvolvedor da *Singa* publicou que também não foi possível replicar os testes exibidos pela *H2O* (NG et al., 2016). Percebe-se, a partir de situações como estas, a importância de se produzir comparativos entre as plataformas de *Deep Learnig* existentes, buscando identificar as características predominantes de cada uma delas.

Como já mencionado, esse trabalho compara a plataforma *Singa* e a plataforma *H2O*. A comparação foi realizada testando métricas como a acurácia e o tempo necessário para treinamento da rede. Para execução dos testes foi utilizada a base de dados *MNIST* (LECUN; CORTES, 2010).

Nos resultados, (NG et al., 2016) traz tabelas como na Figura 4.4, onde o objetivo é mostrar algumas diferenças entre as duas plataformas. Também traz gráficos como na Figura 4.5, que mostram os resultados de acurácia quando as diferentes plataformas foram testadas com a base *MNIST*. Os valores do percentual de acurácia da Figura 4.5 foram obtidos tirando-se amostras a cada 40 minutos de intervalo durante o treinamento. Sendo assim, com os dados desta figura podemos observar que quando a rede passa por um período curto de treinamento, por exemplo 40 minutos, a plataforma *Singa* se mostra superior. Mas se a rede passar por um treinamento de 160 minutos, a plataforma *H2O* começa a mostrar resultados melhores.

Em sua conclusão, esse trabalho mostra que a plataforma *H2O* obteve uma performance estável e bem acurada para a base de dados *MNIST*. E por outro lado, *Singa* mostrou ter um bom desempenho quando a rede é treinada em um curto período de tempo, embora a acurácia varie bastante além do esperado quando detalhes de treinamento são alterados. Como os mesmos algoritmos e a mesma base de dados foi utilizada para os testes em ambas as plataformas, a diferença de resultados de acurácia e performance se devem principalmente à diferenças de implementação. Isso porque o efeito gerado pelo fato do conjunto de treinamento ser randômico

Attributes	H2O	SINGA
Direction	Machine Learning	Deep Learning
Native Language	Java	C++
Supported Languages	Java, Python, R, Scala etc.	C++
User Interface	Web UI	Command Line Interface
Training Framework	Hogwild	Sandblaster, All Reduce, Downpour, Distributed Hogwild
Model/Algorithm	Deep Learning, Generalized linear model (GLM), Decision Tree, Gradient Boosting, K-Means, Anomaly detection, Naïve Bayes, Grid Search	Deep Learning models including convolutional neural networks (CNN), restricted Boltzmann machine (RBM), and recurrent neural networks (RNN)

Figura 4.4 – Diferenças entre as plataformas H2O e Singa

Fonte: (NG et al., 2016)

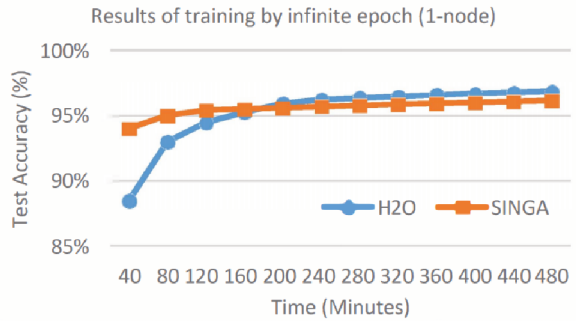

Figura 4.5 – Diferenças entre as plataformas H2O e Singa

Fonte: (NG et al., 2016)

Trabalho	Ferramentas Comparadas
(BAHRAMPOUR et al., 2015)	Caffe, Neon, TensorFlow, Theano e Torch
(KOVALEV; KALINOVSKY; KOVALEV, 2016)	Theano, Torch, Caffe, TensorFlow e Deeplearning4J
(NG et al., 2016)	Apache Singa e H2O
O presente trabalho	Apache Singa, Graphlab e H2O

Tabela 4.1 – Características dos trabalhos relacionados e do presente trabalho

é pequeno, esse trabalho utilizou-se do teste T pareado ou *paired-t-test* para confirmar isso (esse teste pareado gera a média da diferença entre duas amostras independentes).

Por fim, esse trabalho também aconselha quem está buscando uma plataforma para implementação de um projeto de *Deep learning*. No caso de ter pouco tempo para treinamento, a sugestão é usar *Singa*. Caso exista tempo para treinamento e *hardware* disponível, *H2O* mostra uma melhor performance. Outro ponto de destaque é que a implementação da plataforma *Singa* é menos robusta, quando se trata dos detalhes de treinamento, de modo que os usuários precisam configurar os parâmetros com mais cautela ao criar um projeto.

Como pode-se observar na Tabela 4.1, os trabalhos de (BAHRAMPOUR et al., 2015) e (KOVALEV; KALINOVSKY; KOVALEV, 2016) tem seu foco na comparação de *frameworks*. Assim, o trabalho relacionado mais próximo ao presente trabalho é o de (NG et al., 2016), que faz a comparação de duas plataformas de *Deep learning*.

5 EXPERIMENTOS

Neste capítulo serão apresentadas as bases de dados utilizadas, a configuração do ambiente de execução, as métricas a serem analisadas e os experimentos de avaliação das plataformas *Graphlab*, *H2O* e *Singa*. Para cada experimento será apresentada a configuração utilizada em cada plataforma, bem como os resultados obtidos.

5.1 Bases de dados

Para obter resultados mais generalizados, optou-se por fazer a utilização de duas bases de dados com princípios diferentes. Em uma delas, cada exemplo é composto por uma imagem e o seu rótulo. Na outra, os exemplos são compostos por atributos numéricos e textuais mais o rótulo de cada exemplo. Ambas as bases de dados podem ser utilizadas, então, para treinamentos do tipo supervisionado. Essas bases serão apresentadas a seguir.

5.1.1 MNIST

Essa é uma base de dados com dígitos escritos a mão, disponíveis no formato de imagem. Hoje é largamente utilizada para comparar diferentes abordagens de ML para classificação de imagem. Ela possui um conjunto de 60 mil exemplos para treinamento e 10 mil exemplos para teste. Para esta base o objetivo é rotular uma entrada para seu respectivo dígito representante (LECUN; CORTES, 2010). Existem 10 possíveis rótulos, que representam os dígitos de 0 a 9. Estes conjuntos de dados estão disponíveis gratuitamente para *download* no formato binário. Na Figura 5.1 pode ser observado como são os exemplos da base MNIST.

Para facilitar o uso da base MNIST nas ferramentas, os dados de treinamento e teste foram convertidos para o formato CSV. Assim cada exemplo, que é formado por uma imagem de 28x28 pixeis e seu respectivo rótulo, gerou uma linha no arquivo CSV com 784 colunas (uma para cada pixel 28x28) mais uma coluna para o rótulo. O arquivo CSV de treinamento utilizado ficou no tamanho de 109.581.378 *bytes*. E, o arquivo para testes no tamanho de 18.294.827 *bytes*.

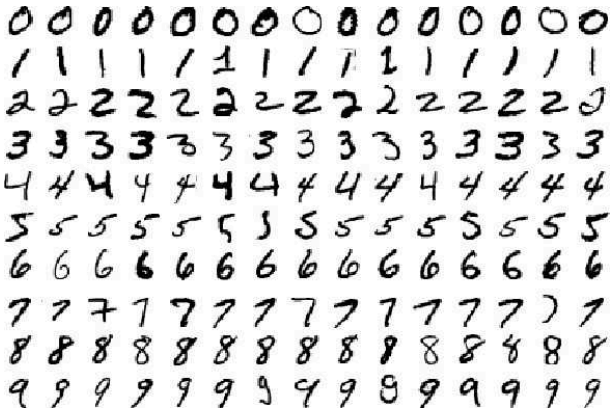

Figura 5.1 – Exemplos de imagens da base MNIST

Fonte: (LECUN; CORTES, 2010)

5.1.2 KDD Cup 1999

Diferentemente da base MNIST, essa base não possui dados que representam imagens. A base KDD foi utilizada na Terceira Competição Internacional de Ferramentas de Descoberta de Conhecimento e Ferramentas de Mineração de Dados (*The Third International Knowledge Discovery and Data Mining Tools Competition*), realizada em conjunto com a Quinta Conferência Internacional sobre Descoberta de Conhecimento e Mineração de Dados (*The Fifth International Conference on Knowledge Discovery and Data Mining*). A tarefa da competição era construir um detector de intrusão de rede. Um modelo preditivo capaz de distinguir entre conexões "ruins", chamadas intrusões ou ataques, e conexões normais "boas".

A base KDD está disponível gratuitamente para *download* no site UCI (DHEERU; KARRA TANISKIDOU, 2017). Ela possui variações de conjuntos para serem utilizados, onde é possível obter parcialmente a base de dados ou também os conjuntos com os dados não rotulados. Para os treinamentos dos experimentos, que serão apresentados a seguir, foi utilizado o conjunto rotulado completo com 4.898.431 instâncias (arquivo no formato CSV com 742.579.829 *bytes*). Além do conjunto rotulado para testes, com 311.029 instâncias (arquivo no formato CSV com 47.251.540 *bytes*).

Esta base é formada por exemplos com 31 atributos que estão descritos na Tabela 5.1. Além destes, existem mais 10 atributos derivados dos descritos na tabela. Assim a base possui um total de 41 atributos. Cada um dos exemplos é rotulado para um dos seguintes tipos conexões: *land*, *pod*, *satan*, teardrop, *ipsweep*, *warezclient*, *normal*, *ftp_write*, *rootkit*, *spy*, *guess_passwd*, *buffer_overflow*, *loadmodule*, *smurf*, *phf*, *imap*, *nmap*, *perl*, *back*, *multihop*,

portsweep, *neptune*, *warezmaster*. Todos esses tipos, exceto o *normal*, são considerados "ruins". Uma conexão rotulada como *normal* é uma conexão "boa", que não representa um ataque.

5.2 Métricas

Com o objetivo de obter resultados comparáveis entre as plataformas *Graphlab*, *H2O* e *Singa* foram definidas quatro métricas apresentadas a seguir:

- **Acurácia:** nesta métrica é analisada a acurácia do modelo gerado pela plataforma. Com isso é possível analisar e discutir sobre como cada plataforma se adapta às diferentes configurações que lhes são submetidas. Também é possível verificar a capacidade que cada plataforma tem para trabalhar com diferentes tipos de dados. Para fazer a medição da acurácia, foi utilizado o conjunto de testes de cada uma das bases de dados apresentadas na seção 5.1. Assim é possível determinar a porcentagem de acurácia que o modelo obteve fazendo a predição dos rótulos para cada exemplo dos conjuntos de testes;

- **Treinamento:** com esta métrica, medida em segundos, é possível fazer uma análise quanto ao desempenho das plataformas para o tempo de treinamento. Para medir o tempo necessário para o treinamento em cada situação, foi utilizado o pacote *time* disponível na linguagem *Python*;

- **Predição:** após ser finalizado o treinamento do modelo de *Deep learning*, as ferramentas permitem fazer predições. Assim, nesta métrica, foi analisado o tempo, em segundos, que cada plataforma leva para rotular um conjunto de itens de teste. Também utilizou-se o pacote *time* da linguagem *Python*;

- **Memória:** para esta métrica é feito o registro do pico de memória consumida, em MB, pela plataforma durante a execução do treinamento. Assim é possível verificar a eficácia da plataforma em trabalhar em ambientes onde os recursos são mais limitados. Para medir o consumo máximo de memória RAM consumida pelo processo em execução utilizou-se a ferramenta *time* do *Linux*;

- **Linhas:** a última métrica a ser avaliada é a dificuldade em fazer a utilização de cada plataforma. Neste item é contabilizado o número de linhas de código que precisaram ser escritas para cada experimento. Neste item foram desconsideradas linhas em "branco" e

Atributo	Tipo	Definição
duration	Contínuo	Número de segundos da conexão
protocol_type	Discreto	Tipo do protocolo (tcp, udp . . .)
service	Discreto	Serviço utilizado (http, telnet . . .)
flag	Discreto	Normal ou erro na conexão
src_bytes	Contínuo	Número de *bytes* da origem para o destino
dst_bytes	Contínuo	Número de *bytes* do destino para a origem
land	Discreto	1 se a conexão é do mesmo *host*/porta, senão 0
wrong_fragment	Contínuo	Número de fragmentos "errados"
urgent	Contínuo	Número de pacotes urgentes
hot	Contínuo	Número de indicadores "*hot*"
num_failed_logins	Contínuo	Número de tentativas de *login* sem sucesso
logged_in	Discreto	1 se estiver *logado* com sucesso, senão 0
num_compromised	Contínuo	Número de condições "comprometidas"
root_shell	Contínuo	1 se o "shell" foi obtido com *root*, senão 0
su_attempted	Contínuo	1 se existe tentativa de "su root", senão 0
num_root	Contínuo	Número de acessos "root"
num_file_creations	Contínuo	Número de operações de criação de arquivo
num_shells	Contínuo	Número de *prompts shell*
num_access_files	Contínuo	Número de operações de acesso à arquivos de controle
num_outbound_cmds	Contínuo	Número de comandos de saída em uma sessão *ftp*
is_host_login	Discreto	1 se se o login pertence à lista *"hot"*, senão 0
is_guest_login	Discreto	1 se o *login* e um *login guest*, senão 0
count	Contínuo	Número de conexões para o mesmo *host* da conexão atual nos últimos 2 segundos
srv_count	Contínuo	Número de conexões para o mesmo serviço da conexão atual nos últimos 2 segundos, para conexões de mesmo *host*
serror_rate	Contínuo	Porcentagem de conexões que possuem erros *SYN*, para conexões de mesmo *host*
srv_serror_rate	Contínuo	Porcentagem de conexões que possuem erros *SYN*, para conexões de mesmo serviço
rerror_rate	Contínuo	Porcentagem de conexões que possuem erros *REJ*, para conexões de mesmo *host*
srv_rerror_rate	Contínuo	Porcentagem de conexões que possuem erros *SYN*, para conexões de mesmo serviço
same_srv_rate	Contínuo	Porcentagem de conexões para o mesmo serviço, para conexões de mesmo *host*
diff_srv_rate	Contínuo	Porcentagem de conexões para diferentes serviços, para conexões de mesmo *host*
srv_diff_host_rate	Contínuo	Porcentagem de conexões para diferentes *hosts*, para conexões de mesmo serviço

Tabela 5.1 – Descrição de atributos da base de dados KDD

Fonte: (DHEERU; KARRA TANISKIDOU, 2017)

comentários. Também foi mantido um mesmo padrão de código para produzir a execução nas três plataformas.

Para validar as comparações entre as plataformas, em cada métrica, foi utilizado o teste *Student's T-Test* (*t-test*). Esse teste é feito sobre dois conjuntos de dados e o seu resultado é um número, entre 0 e 1, que mede a confiança de uma afirmação. Neste trabalho, as afirmações que são passíveis de validação são de que uma plataforma obteve um melhor resultado do que a outra em determinado teste. Assim, a Equação 5.1 traz as hipóteses que serão levantadas nas seções dos experimentos. Na hipótese H_1, desta equação, tem-se que as duas plataformas comparadas são iguais caso o resultado do *t-test* for $\alpha > 0,05$. E, na hipótese H_2 desta mesma equação, pode-se afirmar que uma plataforma foi melhor ou pior que a outra em um determinado aspecto.

$$\begin{cases} H_1 : Plataforma\ X_1 = Plataforma\ X_2, & \text{se } \alpha > 0,05 \\ H_2 : Plataforma\ X_1 \neq Plataforma\ X_2, & \text{se } \alpha < 0,05 \end{cases} \tag{5.1}$$

Em resumo, como o valor de α foi definido em $0,05$, ao validar-se a hipótese H_2 da Equação 5.1, pode-se afirmar com 95% de confiança que a plataforma X_1 obteve um resultado médio diferente da plataforma X_2.

5.3 Configuração do ambiente de testes

Para a execução dos experimentos, foi utilizada uma única máquina com sistema operacional Ubuntu Linux 16.04 (64 bits). Essa máquina possui 8GB de memória *RAM*, um processador *Intel Core i5 - 3470 @ 3,20 GHz* e um *HD* de 1TB - 7200 *RPM*. Nesta máquina foram instaladas as plataformas *Graphlab* (*v2.1*), *H2O* (*v3.18.0.11*) e *Singa* (*v1.2.0*) seguindo-se a documentação disponível. Todos os algoritmos criados nas três plataformas foram desenvolvidos na linguagem de programação Python (*Python 2.7.0*). Em todos os experimentos descritos a seguir, os treinamentos foram executados na *CPU* do computador. Não foram realizados testes fazendo a utilização de *GPU*.

5.4 Experimento 1: Rede recomendada

Neste primeiro experimento, cada uma das plataformas recebeu apenas a base de dados MNIST. Como a base MNIST é largamente utilizada em estudos na área de ML, as plataformas já oferecem configurações recomendadas para executar o aprendizado sobre a base MNIST.

Diferente disso, por ser uma base menos conhecida e as plataformas não trazerem configurações recomendadas para esse tipo de abordagem, a KDD não foi incluída neste experimento.

Para obter-se valores mais fidedignos para a comparação entre as plataformas, cada teste foi executado 10 vezes. Portanto, ao executar o modelo de *Deep learning* recomendado para a base MNIST, em cada uma das plataformas, a Tabela 5.2 foi gerada. Nela, já é possível perceber as diferenças entre as plataformas observando-se os dados.

Após coletar os dados de cada execução, a Tabela 5.3 apresenta os resultados finais. Ou seja, as médias entre todas as execução realizadas para cada plataforma, ao utilizar o modelo recomendado para a base MNIST. Nela também está incluída a coluna com a quantidade de linhas de código que foi necessário, que mede a quantidade de código que foi necessário escrever para executar os teste. Na Tabela 5.3 também foram destacados, em negrito, os melhores resultados. Assim, observa-se que a plataforma *Singa* não teve destaque em nenhuma das métricas avaliadas. Já a plataforma *H2O* teve destaque com os menores tempos para treinamento e predição. Nas avaliações de acurácia, utilização de memória e quantidade de linhas, a plataforma *Graphlab* se mostrou melhor.

As afirmações acima, em que uma plataforma obteve melhor resultado que outra em determinado aspecto, podem ser consideradas com 95% de confiança, pois todos os resultados do *t-test* foram $\alpha < 0,05$. Na Tabela 5.4 pode-se verificar que todas as comparações validam a hipótese H_2 da Equação 5.1.

5.5 Experimento 2: Rede automática

As plataformas *Graphlab* e *H2O* possuem recursos para fazer a leitura de um conjunto de treinamento, extrair informações e a partir disso, sugerir uma rede neural de *Deep learning* automaticamente para o problema. Assim, nesta seção, serão apresentado primeiramente os resultados dos modelos sugeridas automaticamente pelas plataformas para a base MNIST. E, em seguida, os resultados dos modelos automáticos para a base KDD.

Neste experimento não foi incluída a plataforma *Singa*, pois ela não oferece recursos para criar um modelo de *Deep learning* automaticamente a partir de um conjunto de dados.

Novamente, neste experimento, os modelos foram executados 10 vezes e os resultados das execuções para a base de dados MNIST com as plataformas *Graphlab* e *H2O* podem ser vistos na Tabela 5.5. Também, para melhorar o entendimento dos resultados foi gerada a Tabela 5.6, que traz as médias das execuções para cada métrica mais a coluna com a quantidade de

Plataforma	Acurácia	Treinamento	Predição	Memória
Graphlab	0,987	483,123	5,365	251,096
	0,985	476,567	4,905	256,552
	0,986	465,670	5,203	252,472
	0,987	465,620	4,960	250,200
	0,987	460,541	4,895	250,200
	0,986	460,374	4,891	252,244
	0,987	471,197	9,994	254,728
	0,988	472,609	5,138	247,892
	0,988	476,380	5,164	253,592
	0,987	465,011	4,888	254,388
H2O	0,902	10,864	0,472	953,508
	0,867	10,139	0,465	928,364
	0,901	10,743	0,458	928,888
	0,908	11,067	0,465	892,952
	0,913	11,183	0,459	957,276
	0,916	11,244	0,251	1.016,03
	0,906	11,316	0,467	891,852
	0,904	12,332	0,249	916,512
	0,906	12,265	0,453	1.012,06
	0,894	11,798	0,478	989,788
Singa	0,976	314,695	1,691	643,220
	0,969	269,615	1,572	643,040
	0,974	312,819	1,600	643,404
	0,976	307,134	1,628	643,380
	0,970	308,716	1,585	643,332
	0,975	313,539	1,620	643,360
	0,975	273,989	1,629	643,248
	0,976	265,086	1,661	643,144
	0,978	277,751	1,570	643,532
	0,975	314,329	1,592	643,240

Tabela 5.2 – Exp. 1: Rede recomendada - Base MNIST

Plataforma	Acurácia		Treinamento		Predição		Memória		Linhas
	Média	STD	Média	STD	Média	STD	Média	STD	
Graphlab	**0,987**	0,0008	469,7	7,52	5,54	1,57	**252,34**	2,6	**51**
H2O	0,901	0,0137	**11,3**	0,68	**0,42**	0,09	948,72	45,4	55
Singa	0,974	0,0026	295,8	21,16	1,62	0,04	643,29	0,14	110

Tabela 5.3 – Exp. 1: Rede recomendada - Base MNIST - Médias finais

Plataformas	Acurácia	Treinamento	Predição	Memória
Graphlab-H2O	8,03E-09	1,48E-17	2,62E-06	3,12E-12
Graphlab-Singa	2,62E-08	4,64E-09	2,39E-05	4,11E-21
H2O-Singa	1,92E-08	1,19E-11	7,23E-11	5,16E-09

Tabela 5.4 – Exp. 1: Rede recomendada - Base MNIST - T-Test

Plataforma	Acurácia	Treinamento	Predição	Memória
Graphlab	0,831	28,874	1,429	1.682,624
	0,828	28,990	1,284	1.680,676
	0,839	28,452	1,254	1.684,440
	0,828	28,457	1,231	1.682,372
	0,831	28,720	1,347	1.684,628
	0,836	28,017	1,238	1.684,240
	0,815	28,423	1,279	1.683,056
	0,827	28,363	1,392	1.680,152
	0,835	27,986	1,268	1.680,916
	0,831	28,776	1,357	1.684,404
H2O	0,969	211,604	0,655	982,188
	0,971	239,428	0,703	909,760
	0,965	156,375	0,867	755,536
	0,965	182,164	0,658	926,936
	0,970	214,713	0,658	958,368
	0,971	224,083	0,650	902,092
	0,967	206,120	0,656	909,708
	0,967	207,972	0,691	836,804
	0,966	183,975	0,665	852,204
	0,971	184,608	0,663	931,096

Tabela 5.5 – Exp. 2: Rede automática - Base MNIST

linhas utilizadas para implementação. Em negrito estão destacados os melhores resultados para cada métrica de avaliação. Assim, pode-se observar que a plataforma *H2O* tem vantagem em fazer um melhor uso da memória durante a execução. Porém *Graphlab* teve uma larga vantagem no tempo de treinamento. No tempo de predição e na acurácia *H2O* mostrou-se expressivamente melhor. E, por final, na quantidade de linhas, *Graphlab* se mostrou com uma maior facilidade no desenvolvimento.

Todas as afirmações do parágrafo anterior foram feitas com 95% de confiança baseando-se no *t-test* realizado. O resultado deste *t-test* pode ser conferido na Tabela 5.7.

Após execução dos testes com a base MNIST, foi realizado o mesmo experimento com a base KDD. Os resultados das dez execuções deste experimento podem ser vistos na Tabela 5.8.

Plataforma	Acurácia		Treinamento		Predição		Memória		Linhas
	Média	STD	Média	STD	Média	STD	Média	STD	
Graphlab	0,830	0,007	**28,51**	0,34	1,31	0,07	1.682,751	1,7	**45**
H2O	**0,968**	0,002	201,1	24,24	**0,69**	0,07	**896,469**	65,9	52

Tabela 5.6 – Exp. 2: Rede automática - Base MNIST - Médias Finais

Plataformas	Acurácia	Treinamento	Predição	Memória
Graphlab-H2O	2,35E-13	3,08E-09	1,68E-08	3,11E-11

Tabela 5.7 – Exp. 2: Rede automática - Base MNIST - T-Test

As médias para cada métrica foram calculadas e adicionadas, juntamente com a quantidade de linhas escritas em cada plataformas para este experimento, à Tabela 5.9. Assim, verifica-se que a plataforma *H2O* obteve um melhor resultado quando a acurácia é levada em consideração. Todavia, *Graphlab* foi melhor em todos os outros pontos avaliados. Neste experimento fica claro que para se obter uma melhor acurácia com a base de dados KDD é preciso treinar o modelo por um tempo considerável.

O teste *Student T-Test* deste experimento, que pode ser visto na Tabela 5.10, comprova as afirmações acima com 95% de confiança.

5.6 Experimento 3: Rede MLP com parâmetros manualmente configurados

Como mencionado no Capítulo 2, MLP é a rede que existe em comum nas três plataformas avaliadas neste trabalho. Sendo assim, este experimento tem por objetivo avaliar as diferenças entre as plataformas quando a mesma estrutura de camadas de uma rede MLP é utilizada.

Como uma rede MLP é formada por camadas totalmente conectadas (*fully-connected*), é preciso configurar quantas serão as camadas da rede neural que compõe o modelo de *Deep learning* a ser utilizado. Sendo assim as três plataformas foram configuradas para ficar com três camadas. A primeira camada, de entrada, com uma unidade para cada atributo da base. Ou seja, para os testes com a base MNIST, a camada de entrada ficou com 784 (28x28) unidades, e para os testes com a base KDD a primeira camada ficou com 41 unidades. A segunda camada (camada oculta), foi configurada com 100 unidades, e com a saída para uma função de ativação *Sigmoid*. Por fim, a terceira camada também difere da base MNIST para a base KDD. Para os testes com a base MNIST, a camada de saída ficou com 10 unidades, que representam os 10

Plataformas	Acurácia	Treinamento	Predição	Memória
Graphlab	0,864	89,776	0,813	1.063,340
	0,810	91,560	0,852	1.040,264
	0,888	89,887	0,861	1.057,184
	0,917	89,277	0,789	1.044,728
	0,907	89,299	0,864	1.045,608
	0,888	89,565	0,840	1.045,460
	0,810	87,038	0,716	1.047,172
	0,809	78,979	0,722	1.046,992
	0,812	80,775	0,719	1.041,200
	0,891	79,892	0,710	1.069,824
H2O	0,926	1.842,648	4,453	1.532,720
	0,922	629,458	5,363	1.469,476
	0,925	2.725,182	4,450	1.585,404
	0,926	1.500,846	3,708	1.509,112
	0,926	1.644,273	4,464	1.542,620
	0,927	1.547,022	3,739	1.561,396
	0,926	1.974,373	4,447	1.521,776
	0,924	891,088	3,763	1.668,884
	0,923	1.378,408	4,451	1.357,408
	0,927	2.392,586	4,446	1.439,208

Tabela 5.8 – Exp. 2: Rede automática - Base KDD

Plataforma	Acurácia		Treinamento		Predição		Memória		Linhas
	Média	STD	Média	STD	Média	STD	Média	STD	
Graphlab	0,860	0,045	**86,61**	4,783	**0,79**	0,07	**1.050,2**	9,9	**45**
H2O	**0,925**	0,002	1.652,6	630,2	4,33	0,5	1.518,8	84,6	52

Tabela 5.9 – Exp. 2: Rede automática - Base KDD - Médias Finais

Plataformas	Acurácia	Treinamento	Predição	Memória
Graphlab-H2O	1,00E-03	2,55E-05	2,55E-09	2,99E-08

Tabela 5.10 – Exp. 2: Rede automática - Base KDD - T-Test

possíveis rótulos (dígitos de 0 a 9). Já para os testes com a base KDD, a última camada ficou com 2 unidades, que representas os possíveis rótulos (conexão boa ou ruim).

No experimento com a base MNIST cada teste também foi executado 10 vezes, e os resultados estão dispostos na Tabela 5.11. Assim como nos experimentos anteriores, foi gerada a Tabela 5.12 que representa os resultados finais com as médias das execuções. Na tabela com os resultados finais, pode-se perceber que a plataforma *Singa* fez a predição do conjunto de testes em menos tempo. Já a plataforma *Graphlab* se destacou em fazer o treinamento em menos tempo e ter sua implementação realizada com maior facilidade. A plataforma *H2O* teve destaque na acurácia e no consumo de memória, com o melhor desempenho nestes dois aspectos para este experimento.

Utilizando o teste *Student T-Test*, foi gerada a Tabela 5.13. A mesma comprova que as afirmações feitas no parágrafo anterior podem ser tomadas com 95% de confiança, baseando-se nos resultados de cada execução dos testes.

Após analisar os resultados dos testes utilizando a rede MLP nas três plataformas com a base MNIST, foram realizados os testes com as mesmas configurações para a base KDD. Os resultados das 10 execuções em cada plataforma podem ser observados na Tabela 5.14. Para melhor identificar as diferenças entre as plataformas, foi gerada a Tabela 5.15. Nela é possível identificar a superioridade da plataforma *Graphlab* quando memória e quantidade de linhas necessárias são levados em consideração. Já nos aspectos de tempo de treinamento, tempo de predição e na acurácia da predição, a plataforma *H2O* se mostrou melhor. Neste experimento a plataforma *Singa* não obteve destaque em nenhum dos aspectos avaliados.

Novamente, as afirmações acima foram avaliadas utilizando-se o *t-test* e os resultados estão dispostos na Tabela 5.16. Estes resultados mostram que as afirmações do parágrafo anterior podem ser tomadas com 95% de confiança. Uma comparação que não seria válida neste teste é: "*Graphlab* é superior ou inferior à plataforma *Singa* na acurácia". Isso pois o resultado do *t-test* foi $\alpha > 0,05$, o que valida a hipótese H_1 da Equação 5.1.

5.7 Experimento 4: Curvas das métricas para a rede MLP

Neste experimento o objetivo é analisar a curva do tempo de treinamento, acurácia e memória utilizada para cada plataforma com a rede MLP manualmente configurada do experimento anterior. Para isso, cada uma das bases foi utilizada em frações a cada etapa do experimento. Assim, foi possível avaliar a robustez do modelo gerado pela plataforma quando

Plataforma	Acurácia	Treinamento	Predição	Memória
Graphlab	0,940	60,495	1,742	1.909,360
	0,942	60,445	1,783	1.915,040
	0,940	60,167	1,749	1.867,888
	0,938	60,635	1,781	1.928,636
	0,944	60,415	1,739	1.947,960
	0,940	60,206	1,830	1.887,336
	0,940	60,426	1,719	1.976,292
	0,939	60,478	1,732	1.833,604
	0,941	61,167	1,818	1.844,568
	0,941	64,287	2,019	1.768,940
H2O	0,956	136,830	0,682	849,912
	0,953	139,914	0,658	652,072
	0,954	136,940	0,670	656,100
	0,949	145,374	0,663	605,892
	0,956	137,308	0,671	634,280
	0,949	136,787	0,653	669,556
	0,957	135,819	0,664	712,216
	0,952	136,581	0,651	628,316
	0,958	138,090	0,677	792,564
	0,956	140,380	0,668	612456
Singa	0,922	150,332	0,088	2.815,188
	0,920	148,934	0,120	2.815,244
	0,921	135,076	0,123	2.815,241
	0,921	139,094	0,052	2.815,246
	0,921	142,820	0,056	2.815,213
	0,921	134,652	0,056	2.815,248
	0,920	138,358	0,052	2.814,942
	0,922	137,995	0,056	2.815,250
	0,919	133,126	0,048	2.815,569
	0,921	132,258	0,048	2.815,152

Tabela 5.11 – Exp. 3: Rede MLP - Base MNIST

Plataforma	Acurácia		Treinamento		Predição		Memória		Linhas
	Média	STD	Média	STD	Média	STD	Média	STD	
Graphlab	0,940	0,002	**60,9**	1,2	1,79	0,09	1.887,9	61,2	**51**
H2O	**0,954**	0,003	138,4	2,9	0,67	0,01	**681,34**	80,9	55
Singa	0,921	0,001	139,3	6,3	**0,07**	0,03	2.815,2	0,15	110

Tabela 5.12 – Exp. 3: Rede MLP - Base MNIST - Médias finais

Plataformas	Acurácia	Treinamento	Predição	Memória
Graphlab-H2O	8,43E-08	1,42E-14	1,85E-11	1,68E-11
Graphlab-Singa	2,60E-10	4,92E-11	1,28E-12	3,77E-12
H2O-Singa	3,09E-10	7,04E-01	3,57E-13	2,57E-14

Tabela 5.13 – Exp. 3: Rede MLP - Base MNIST - T-Test

Plataforma	Acurácia	Treinamento	Predição	Memória
Graphlab	0,918	510,485	2,657	1.045,924
	0,919	502,003	2,618	1.021,456
	0,919	507,604	2,611	1.024,608
	0,918	498,007	2,544	1.068,408
	0,919	457,849	2,294	1.035,644
	0,917	451,212	2,352	1.044,948
	0,919	447,850	2,266	1.029,464
	0,920	467,121	2,371	1.021,088
	0,919	458,059	2,359	1.025,744
	0,919	466,659	2,364	1.056,736
H2O	0,925	122,682	1,301	1.337,412
	0,926	56,966	1,068	1.357,868
	0,922	98,575	1,304	1.266,560
	0,924	70,713	1,054	1.437,736
	0,925	93,208	1,058	1.499,312
	0,926	68,604	1,082	1.523,924
	0,922	81,332	1,309	1.567,220
	0,927	101,612	1,326	1.416,284
	0,920	84,344	1,139	1.396,156
	0,924	56,693	1,285	1.353,928
Singa	0,918	339,899	1,944	3.062,284
	0,918	349,980	1,921	3.062,400
	0,919	299,379	2,016	3.062,248
	0,919	318,014	1,950	3.062,532
	0,919	320,233	1,948	3.062,296
	0,918	311,898	1,959	3.066,448
	0,918	339,718	1,939	3.062,092
	0,918	390,207	1,912	3.064,344
	0,919	306,459	1,917	3.062,632
	0,916	353,822	1,967	3.062,520

Tabela 5.14 – Exp. 3: Rede MLP - Base KDD

Plataforma	Acurácia		Treinamento		Predição		Memória		Linhas
	Média	STD	Média	STD	Média	STD	Média	STD	
Graphlab	0,919	0,001	476,69	24,9	2,44	0,2	**1.037,4**	16,2	**51**
H2O	**0,924**	0,002	**83,47**	21,1	**1,19**	0,1	1.415,6	93,1	55
Singa	0,918	0,001	332,96	27,4	1,95	0,03	3.062,9	1,38	110

Tabela 5.15 – Exp. 3: Rede MLP - Base KDD - Médias finais

Plataformas	Acurácia	Treinamento	Predição	Memória
Graphlab-H2O	7,74E-05	8,13E-12	5,99E-09	4,23E-07
Graphlab-Singa	1,92E-01	8,11E-07	1,57E-06	2,13E-20
H2O-Singa	4,87E-05	2,52E-09	8,61E-09	8,97E-13

Tabela 5.16 – Exp. 3: Rede MLP - Base KDD - T-Test

diferentes tamanhos de bases são utilizados. Não foram incluídas as métricas de tempo de predição e quantidade de linhas pois elas se mantiveram iguais para as diferentes frações das bases.

Para a execução dos testes, cada conjunto de treinamento foi dividido em 10, 20, 30, 40, 50, 60, 70, 80, 90 e 100 porcento. E, cada execução com uma fração da base foi repetida por 10 vezes. Os resultados apresentados neste experimento são as médias das 10 repetições.

No gráfico da Figura 5.2 pode-se observar as curvas para os tempos de treinamento com a base MNIST. Todas as plataformas demonstraram um crescimento linear no tempo de treinamento, o que é muito importante para bases com grande quantidade de imagens. Também é possível reafirmar o resultado do experimento anterior, onde a plataforma *Graphlab* se mostrou melhor no tempo de treinamento para a base MNIST.

Outro resultado confirmado, com relação ao experimento anterior, em todas as frações da base MNIST é a acurácia superior com a plataforma *H2O*. No gráfico da Figura 5.3, essa

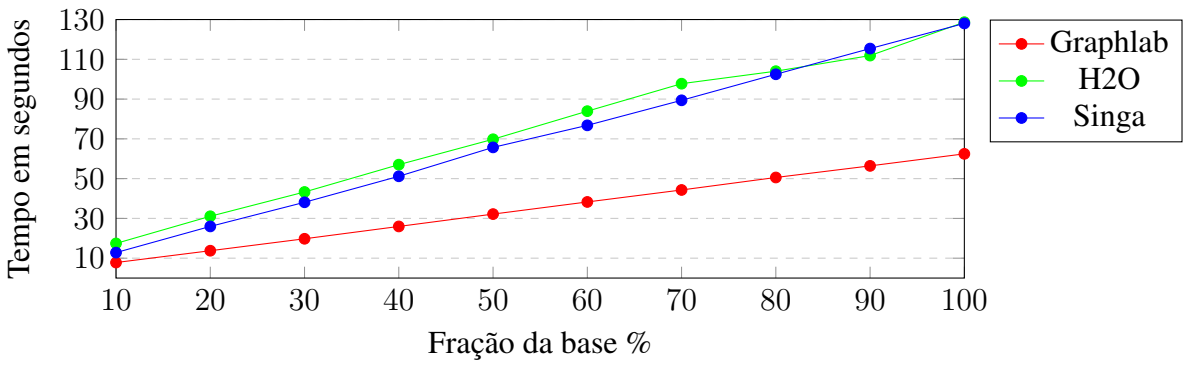

Figura 5.2 – Tempos de treinamento - Base MNIST

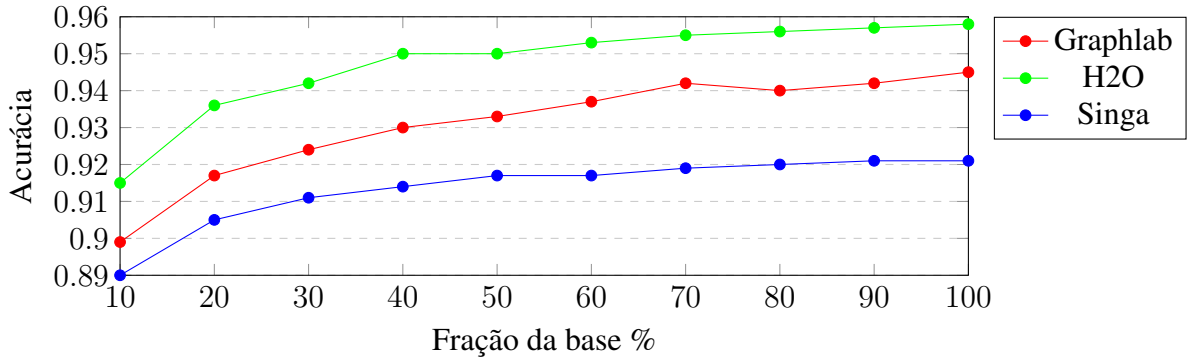

Figura 5.3 – Acurácias - Base MNIST

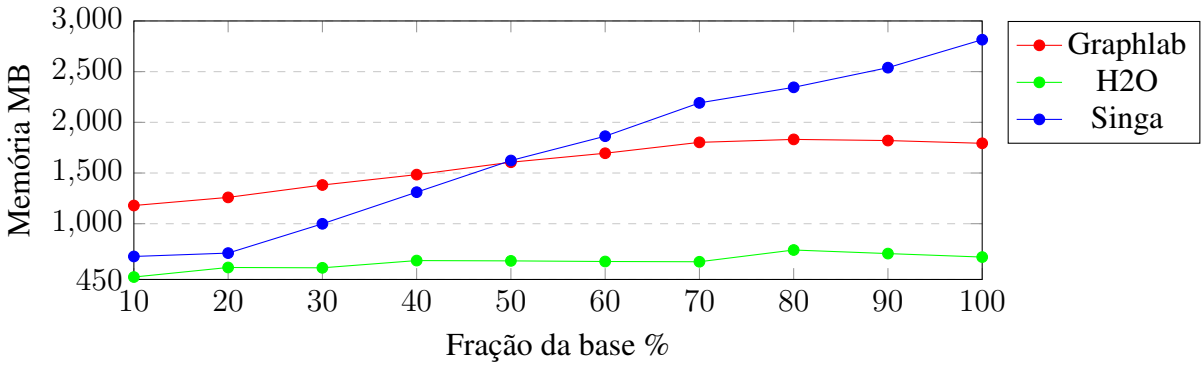

Figura 5.4 – Consumo de memória - Base MNIST

superioridade pode ser observada.

O gráfico da Figura 5.4 apresenta a utilização de memória nas plataformas durante a execução para cada fração da base MNIST. Destaque positivo para a plataforma *H2O* que soube administrar e manter um consumo com pouco crescimento ao aumentar-se o tamanho da base. E destaque negativo para a plataforma *Singa*, que apresenta um aumento no consumo muito expressivo quando a base de treinamento cresce.

Os mesmos testes descritos acima com a base MNIST também foram executados com a base KDD. No gráfico da Figura 5.5 pode-se verificar os resultados do mesmo, onde também reforça-se o resultado obtido no Experimento 3 com a base KDD. *H2O* mostrou-se superior, treinando a rede em menos tempo se comparada às outras duas plataformas.

No gráfico da Figura 5.6 tem-se que a rede MLP utilizada alcança a acurácia com uma pequena fração da base KDD, e depois mesmo aumentando o tamanho do conjunto de treinamento, a acurácia permanece no mesmo patamar. Neste teste houve uma pequena vantagem da plataforma *H2O* em obter um melhor resultado no aspecto acurácia em todas as frações do conjunto de treinamento. Observe também, que a *Graphlab* manteve-se estável na acurácia para todos os lotes de execução.

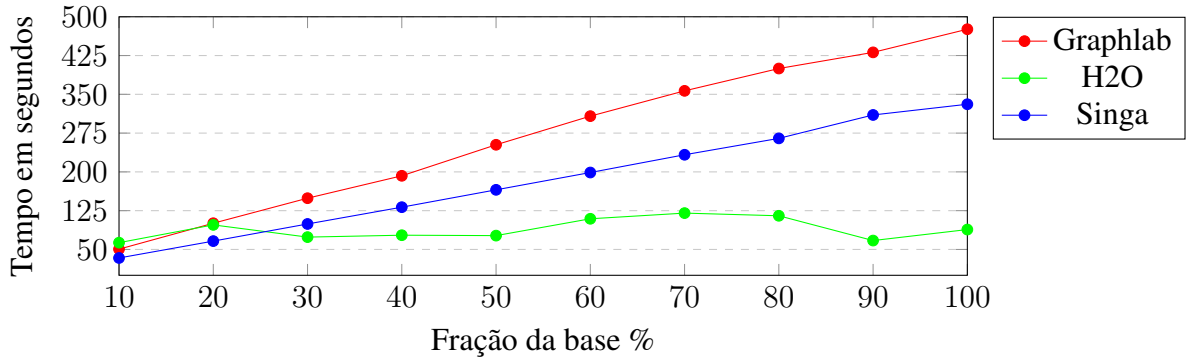

Figura 5.5 – Tempos de treinamento - Base KDD

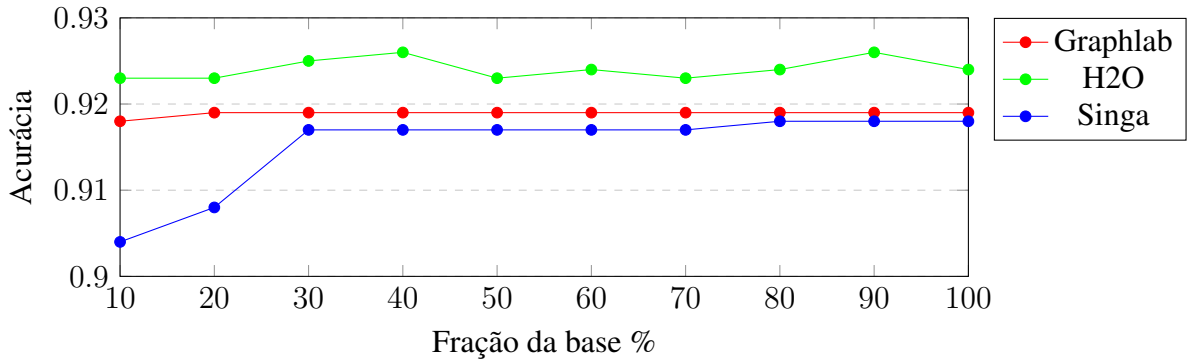

Figura 5.6 – Acurácias - Base KDD

Por fim, o gráfico da Figura 5.7 apresenta a curva do consumo de memória ao variar-se o tamanho do conjunto de treinamento em cada plataforma. Percebe-se que, com as frações menores que 50% do conjunto de treinamento da base KDD, a plataforma *H2O* levou vantagem. Acima de 50% da base o melhor resultado foi da plataforma *Graphlab*. E novamente a plataforma *Singa* teve destaque negativo por fazer a utilização de muita memória ao aumentar-se o conjunto de treinamento.

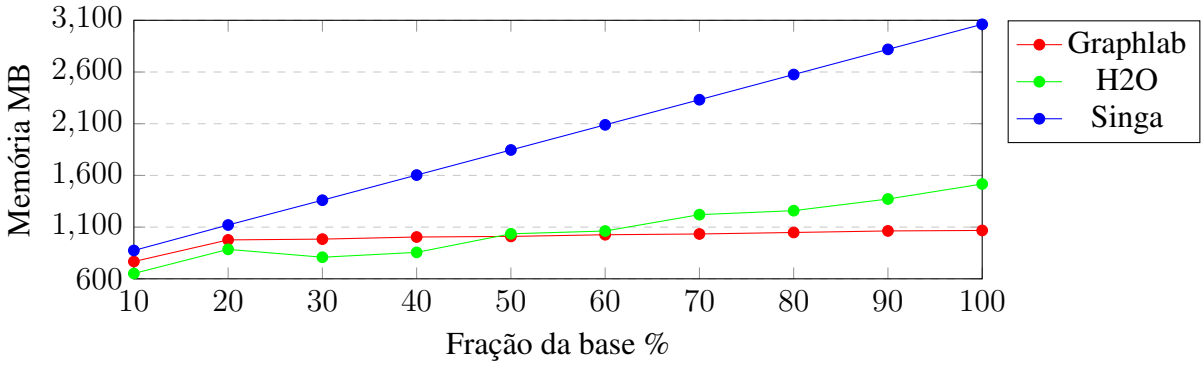

Figura 5.7 – Consumo de memória - Base KDD

5.8 Considerações dos experimentos

Dados os resultados nos experimentos descritos nas seções anteriores, nesta seção cada experimento será retomado e analisado brevemente. Nos itens a seguir, serão apresentadas as plataformas com o melhor resultado para cada teste realizado.

- **Experimento 1:** neste experimento apenas utilizou-se a base MNIST com as configurações recomendadas para cada plataforma. A plataforma *H2O* obteve os melhores tempos, tanto para treinamento como para predição, perdendo no aspecto de acurácia. E a plataforma *Graphlab* teve os melhores resultados para acurácia, memória e quantidade de linhas necessárias para a implementação.

- **Experimento 2:** este foi o experimento onde utilizou-se as configurações automáticas de cada plataforma para ambas as bases. Para a base MNIST, a plataforma *Graphlab* foi melhor em tempo de treinamento e quantidade de linhas. E a *H2O* obteve melhores resultados ao avaliar-se a acurácia, tempo de predição e consumo de memória. Já para a base KDD, a plataforma *H2O* se destacou apenas na acurácia. No restante das métricas avaliadas *Graphlab* foi melhor.

- **Experimento 3:** neste experimento configurou-se uma rede MLP para receber cada uma das bases. Com a base MNIST, a plataforma *Graphlab* mostrou melhor desempenho para tempo de treinamento e quantidade de linhas. Já no aspecto acurácia e consumo de memória, *H2O* foi melhor. *Singa* obteve um tempo menor para a predição com a base MNIST para este experimento. Nos testes com a base KDD, *H2O* foi melhor em acurácia, tempo de treinamento e tempo de predição. E a plataforma *Graphlab* teve resultados mais eficientes no consumo de memória e na quantidade de linhas.

- **Experimento 4:** com este experimento, onde foi utilizada a rede MLP, foi possível perceber que a plataforma *Singa* obteve o pior desempenho no consumo de memória para ambas as bases. As outras duas plataformas demonstraram um consumo mais constante quando uma fração maior do conjunto de treinamento era utilizada. Na avaliação da acurácia, percebeu-se que as três plataformas obtiveram resultados semelhantes, porém com resultados ligeiramente melhores para a plataforma *H2O*. E na avaliação por tempo de treinamento, para a base MNIST a plataforma *Graphlab* se mostrou superior. Já para a

base KDD, a plataforma *H2O* obteve os melhores resultados fazendo o treinamento em menor tempo.

Com os experimentos 1, 2 e 3 foi possível discretizar qual plataforma obteve o melhor desempenho em cada métrica de avaliação. Assim, a Tabela 5.17 foi gerada. Nesta tabela pode ser observado com detalhes onde cada plataforma se destacou.

Experimento	Base	Métrica	1º colocado	2º colocado	3º colocado
Exp. 1	MNIST	Acurácia	Graphlab	Singa	H2O
		Treinamento	H2O	Singa	Graphlab
		Predição	H2O	Singa	Graphlab
		Memória	Graphlab	Singa	H2O
		Linhas	Graphlab	H2O	Singa
Exp. 2	MNIST	Acurácia	H2O	Graphlab	Singa
		Treinamento	Graphlab	H2O	Singa
		Predição	H2O	Graphlab	Singa
		Memória	H2O	Graphlab	Singa
		Linhas	Graphlab	H2O	Singa
	KDD	Acurácia	H2O	Graphlab	Singa
		Treinamento	Graphlab	H2O	Singa
		Predição	Graphlab	H2O	Singa
		Memória	Graphlab	H2O	Singa
		Linhas	Graphlab	H2O	Singa
Exp. 3	MNIST	Acurácia	H2O	Graphlab	Singa
		Treinamento	Graphlab	H2O	Singa
		Predição	Singa	H2O	Graphlab
		Memória	H2O	Graphlab	Singa
		Linhas	Graphlab	H2O	Singa
	KDD	Acurácia	H2O	Graphlab	Singa
		Treinamento	H2O	Singa	Graphlab
		Predição	H2O	Singa	Graphlab
		Memória	Graphlab	H2O	Singa
		Linhas	Graphlab	H2O	Singa

Tabela 5.17 – *Ranking* das plataformas por métrica

6 CONCLUSÃO

Neste trabalho de conclusão de curso foram avaliadas e comparadas três plataformas que possibilitam a construção de modelos de *Deep learning*: *Graphlab*, *H2O* e *Apache Singa*. Para que fosse possível fazer um comparativo entre as plataformas, modelos de *Deep learning* foram criados em cada uma delas avaliando-se métricas como acurácia do modelo, tempo de treinamento, tempo de predição, consumo de memória e quantidade de linhas necessárias para a implementação em cada ferramenta. Diferentes experimentos foram realizados em cima de duas bases de dados: uma com imagens, chamada MNIST e outra com atributos alfanuméricos, chamada KDD.

Depois de realizados os experimentos, foi possível perceber que a plataforma *Graphlab* é a mais simples para ser utilizada por um usuário sem muito conhecimento sobre parametrização e configuração de redes de *Deep learning*. Em oposição à isso, a plataforma *Singa* traz uma proposta para usuários mais experientes, trazendo uma maior flexibilidade de utilização e exigindo um conhecimento maior para ser devidamente configurada.

De forma geral, os melhores resultados de acurácia foram gerados pela plataforma *H2O*. Principalmente quando a plataforma pôde gerar automaticamente o modelo baseando-se no conjunto de dados de treinamento, tanto para a base MNIST como para a base KDD.

Nos testes que levaram em consideração o tempo de treinamento, em sua maioria o destaque foi para a plataforma *Graphlab*. Já no tempo de predição, a plataforma *H2O* desempenhou a execução preditiva do conjunto de testes com o menor tempo na maioria dos experimentos.

Quanto ao consumo de memória, ao utilizar-se a base MNIST, composta por imagens, o melhor desempenho foi da plataforma *H2O*. E, nos testes com a base KDD que é composta por atributos alfanuméricos, a plataforma *Graphlab* conseguiu realizar o processo de treinamento consumindo a menor quantidade de memória.

6.1 Trabalhos futuros

Uma forma de abranger o estudo comparativo realizado neste trabalho é executar os experimentos com a utilização de uma *GPU* (placa gráfica). Hoje, nos computadores, a maioria das placas gráficas possuem um desempenho superior com relação às *CPUs* para executar algoritmos de *Deep learning*. Além disso, as plataformas também têm suporte para a execução dos algoritmos na *GPU*. Assim este tipo de experimento complementaria os resultados deste

trabalho.

Outro trabalho importante seria testar os modelos gerados automaticamente pelas plataformas com várias bases de dados diferentes. Uma vez que exige menos conhecimento para ser utilizado, esse procedimento automático pode beneficiar uma grande quantidade de usuários menos experientes.

Por fim, outra perspectiva futura é executar testes com treinamento de *Deep learning* de modo distribuído. Ou seja, fazendo os experimentos com a utilização em mais de uma máquina com o objetivo de obter-se um melhor desempenho no tempo de treinamento.

REFERÊNCIAS

AREL, I.; ROSE, D. C.; KARNOWSKI, T. P. Deep machine learning-a new frontier in artificial intelligence research [research frontier]. **IEEE computational intelligence magazine**, [S.l.], v.5, n.4, p.13–18, 2010.

BAHRAMPOUR, S. et al. Comparative Study of Deep Learning Software Frameworks. **arXiv preprint arXiv:1511.06435**, [S.l.], 2015.

CANDEL, A. et al. **Deep learning with h2o**. [S.l.]: H2O, 2015.

DEAN, J. et al. Large scale distributed deep networks. In: ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS. **Anais...** [S.l.: s.n.], 2012. p.1223–1231.

DENG, J. et al. Imagenet: a large-scale hierarchical image database. In: COMPUTER VISION AND PATTERN RECOGNITION, 2009. CVPR 2009. IEEE CONFERENCE ON. **Anais...** [S.l.: s.n.], 2009. p.248–255.

DENG, L.; YU, D. Deep learning: methods and applications. **Foundations and Trends® in Signal Processing**, [S.l.], v.7, n.3–4, p.197–387, 2014.

DHEERU, D.; KARRA TANISKIDOU, E. **UCI Machine Learning Repository**. 2017.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep Learning**. [S.l.]: MIT Press, 2016. http://www.deeplearningbook.org.

KOVALEV, V.; KALINOVSKY, A.; KOVALEV, S. Deep Learning with Theano, Torch, Caffe, Tensorflow, and Deeplearning4J: which one is the best in speed and accuracy? **None**, [S.l.], 2016.

LECUN, Y.; BENGIO, Y.; HINTON, G. Deep learning. **Nature**, [S.l.], v.521, n.7553, p.436–444, 2015.

LECUN, Y.; CORTES, C. **MNIST handwritten digit database**. 2010.

LOW, Y. et al. Distributed GraphLab: a framework for machine learning and data mining in the cloud. **Proceedings of the VLDB Endowment**, [S.l.], v.5, n.8, p.716–727, 2012.

LOW, Y. et al. Graphlab: a new framework for parallel machine learning. **arXiv preprint arXiv:1408.2041**, [S.l.], 2014.

MAAS, A. L. et al. Learning Word Vectors for Sentiment Analysis. In: ANNUAL MEETING OF THE ASSOCIATION FOR COMPUTATIONAL LINGUISTICS: HUMAN LANGUAGE TECHNOLOGIES, 49., Portland, Oregon, USA. **Proceedings...** Association for Computational Linguistics, 2011. p.142–150.

MITCHELL, T. M. **Machine Learning**. 1.ed. New York, NY, USA: McGraw-Hill, Inc., 1997.

NG, S. et al. An independent study of two deep learning platforms-H2O and SINGA. In: INDUSTRIAL ENGINEERING AND ENGINEERING MANAGEMENT (IEEM), 2016 IEEE INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2016. p.1279–1283.

OOI, B. C. et al. SINGA: a distributed deep learning platform. In: ACM MULTIMEDIA. **Anais...** [S.l.: s.n.], 2015.

SHAMS, S. et al. Evaluation of Deep Learning Frameworks Over Different HPC Architectures. In: IEEE 37TH INTERNATIONAL CONFERENCE ON DISTRIBUTED COMPUTING SYSTEMS (ICDCS), 2017. **Anais...** [S.l.: s.n.], 2017. p.1389–1396.

UETZ, R.; BEHNKE, S. Large-scale object recognition with CUDA-accelerated hierarchical neural networks. In: INTELLIGENT COMPUTING AND INTELLIGENT SYSTEMS, 2009. ICIS 2009. IEEE INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2009. v.1, p.536–541.

WANG, W. et al. SINGA: putting deep learning in the hands of multimedia users. In: ACM MULTIMEDIA. **Anais...** [S.l.: s.n.], 2015.